Rhinoceros×Python

コンピュテーショナル・デザイン入門

三井和男 著

彰国社

Rhino-ceros×Python

コンピュテーショナル・デザイン入門

三井和男 著

彰国社

はじめに

三井和男

私たちはものを作るのが大好きです。人と動物を隔てるものは道具を使うことであると言われますが、有史以前から人間は道具を、ものを作ってきました。遠い昔、私たちの祖先は石器や土器などの道具を作りました。その道具を使って身につけるものを作り、煮炊きをして食べるものを作りました。道具を使ってもっと複雑な道具も作ってきました。今ではとてつもなく複雑な機械も作っています。私たちは基本的にものを作るのが好きなのです。

近年、ものづくりの大好きな私たちには歓迎すべき大きな変化が起こっています。3Dプリンターやレーザーカッター、CNC装置、3Dスキャナーなどの登場です。21世紀の工房にはこれらデジタルファブリケーション機器と呼ばれるツールが続々と登場し、新産業革命と呼ばれるものづくりの新しい未来を照らし出しています。これらの機器並んで歓迎すべきもう一つは、SOLIDWORKS、Rhinoceros、Fusion360などのデザインツールです。これらのツールは、頭の中にあるアイディアをマウスとキーボードを使ったデスクトップの作業でドキュメントとして作りあげます。CAD（Computer Aided DesignもしくはDrawingの略）と総称されるこれらのツールの中には、アイディアをドキュメントに、そしてシミュレーションに連携して、製造工程で必要なデータにまでまとめあげるものもあります。デジタルファブリケーション機器とCADのようなデザインツールは、情報を物質に、そしてまた物質を情報に変換する魔法の杖といえるでしょう。

CADの先駆けは、1963年に発表されたSketchpadとされています。グラフィカルユーザインタフェースを実現し、それまでのトレーシングペーパーに定規と鉛筆という人間の作業をコンピュータに置き換えました。その後、コンピュータ支援設計というよりは、コンピュータを用いた製図システムという方が適切な時代が続きましたが、今ではこの状況も大きく変化しています。GrasshopperのようなGAE（Graphic Algorithm Editor）の登場です。GAEを使うと設計のプロセスを直感的なアイコンで表現されるコンポーネントの接続によって記述できます。GAEが単なる3Dモデラーと異なるのは、プロセスモデルを記述することによって、パラメータを変更したり、部分的なプロセスを組み替えたり修正したりすることでアウトプットを理想の精度にまで高めるということを可能にする点にあります。しかし、GAEでは使えるコンポーネントの種類に限りがあって、やりたいことを記述するには非常に回りくどいコンポーネントの接続が必要であったり、あるいは適切なコンポーネントが存在しない場合さえあります。設計の自由度を高めるためには、コンポーネントに頼らず自分でスクリプトを書かなければならない場合も多いでしょう。Pythonは、それを実現するプログラミング言語の一つです。また、PythonによってGrasshopperのコンポーネントを構築することも可能ですから、自由度を高めるために、Pythonによる設計アルゴリズムの記述の必要性は一層増すと考えられます。

この本は、Chapter 1からChapter 3までの構成になっています。Chapter 1では、Pythonプログラミング言語の基礎を学びます。Pythonを学んだことのない読者を対象に、この本で必要な基礎知識をRhino Python Editorを使って解説します。Pythonをご存知の方は、読み飛ばすことが可能です。Chapter 2は、RhinoとPythonを結びつけるところです。RhinoのコマンドをPythonのスクリプトからどのようにして実行するのか、また、どのようなコマンドがあるのかについて知ることができるでしょう。Chapter 3では、いよいよ計算によって形を表現します。どのように考え、そしてそのプロセスをどのように記述するかについて知ることができるでしょう。Chapter 3の1つ目は、サボテンを描きます。おそらく誰もが知っているイメージをそのまま記述してみます。2つ目は、巻貝の形状を記述します。イメージを数式で表現し、それをコードで記述します。3つ目では、再帰アルゴリズムを扱います。プログラミングに特有な手法の一つですが、自然界のさまざまな現象と再帰との興味深い関係を知ることができるでしょう。4つ目はワッフリングというモデリングの一手法を扱います。5つ目では、最適化アルゴリズムで形を見つけることに挑戦します。6つ目では、反応拡散方程式を解くことによってチューリング・パターンを生成します。

楽しみながらプログラミングの学習を進め、プログラミングをデザインという活動の一つの手段、または発想の手がかりとして活用していただけるよう、また、ものづくりの自由度を高めていただけるよう願います。

はじめに　三井和男　**002**
この本の使い方　**007**

準備編　Rhinoceros と Python のセットアップ　009

1　Rhinoceros と Python の関係を知る　**010**
2　Python のエディタを起動する　**011**
3　プログラム・コードを書く　**012**
4　デバッグする　**013**
Rhinoceros for Mac で Python スクリプトを始める方法　**014**

Chapter 1　Python プログラミングの基礎　015

1　ビルトインオブジェクト　**016**
数値／文字列／リスト／ディクショナリ／タプル／ブーリアン
2　変数　**019**
変数の役割／名前の付け方／変数とタグ
3　オブジェクトの操作　**021**
数値の操作／文字列の操作／リストの操作／ディクショナリの操作
4　ステートメント　**034**
代入ステートメント／ if ステートメント／ while ステートメント／ for ステートメント／ print ステートメント
5　関数　**042**
関数の定義／関数の呼び出し
6　オブジェクト指向プログラミング　**045**
クラスとインスタンス／サブクラス

Chapter 2　Rhinoceros × Python　049

1　点　**050**
点の情報を取得する／点を生成する／点を円周に沿って配置する／点をスパイラル状に配置する
2　曲線　**056**
曲線の情報を取得する／曲線を生成する／曲線を修正する／曲線の接線と法線を計算する
3　曲面　**065**
曲面の情報を取得する／基本的な立体を生成する／4点を指定して曲面を生成する／エッジを指定して曲面を生成する／点群から曲面を生成する／回転曲面を生成する／ロフトで曲面を生成する／曲面の法線を描く
4　メッシュ　**076**
メッシュで曲面を表現する／関数を使ってメッシュを生成する
5　ベクトル　**081**
ベクトルを矢印で描く／ベクトル成分を使って描く／ベクトルを描くための関数を作る／ベクトル演算を実行する／曲線の接線と法線を計算する
6　ブーリアン　**090**
曲線と曲線の交点を計算する／ 2 つの図形の共通部分を計算する／閉曲線を結合する

Chapter 3　コンピュテーショナル・デザイン　095

1　イメージした形とプログラム　**096**
プログラミングのアウトライン／ Step 1　パラメータを設定する／ Step 2　尾根曲線を決める／ Step 3　尾根を配置する／ Step 4　Loft のための断面曲線を描く／ Step 5　尾根曲線をもとに断面曲線を描く／ Step 6　ロフトしてサボテンの本体を作る／ Step 7　棘の基本形を作る／ Step 8　棘を配置する
2　数式と貝殻　**114**
プログラミングのアウトライン／ Step 1　平面上に螺旋を描く／ Step 2　螺旋のパラメータを決定する／ Step 3　Loft を使って貝殻の曲面を作る／ Step 4　円錐の表面に巻き付いた螺旋を考える／ Step 5　Loft のための断面を生成する

3　再帰アルゴリズム　124

再帰アルゴリズムを試す／プログラミングのアウトライン／ Step 1　幹を描く／ Step 2　最初の枝分かれを作る／ Step 3　もう片方に伸びる枝分かれも作る／ Step 4　それより先に伸びる枝分かれを作る／ Step 5　再帰で繰り返す／ Step 6　線で描いた樹木を立体にする

4　ワッフリング　134

プログラミングのアウトライン／ Step 1　ポリサーフェスに変換する／ Step 2　縦材の輪郭線を描く／ Step 3　輪郭線から縦材の面を作る／ Step 4　横材の輪郭線を描く／ Step 5　輪郭線から横材の面を作る

5　最適化アルゴリズム　144

プログラミングのアウトライン／ホタルのクラスを設計する／ Step 1　モジュールをインポートしてパラメータを設定する／ Step 2　クラスの見出しと初期化メソッドを書く／ Step 3　目的関数値を計算する／ Step 4　ホタルの移動を計算する／ Step 5　ホタルを表示する／ Step 6　最適化の例題へ適用して試す／ Step 7　極小曲面問題に挑戦する／ Step 8　枠を作って準備する／ Step 9　枠に膜を張る／ Step 10　極小曲面問題へ適用する／ Step 11　曲面の情報を取得する／ Step 12　評価値を計算する／ Step 13　曲面を描く／ Step 14　解を見つけるための準備をする／ Step 15　ホタルの群れを初期化する／ Step 16　探索を開始する／ Step 17　ホタルの群れから最も優れた解を探し出す

6　チューリング・パターン　164

反応拡散方程式／プログラミングのアウトライン／ Step 1　定数を設定する／ Step 2　2次元のリストを初期化する／ Step 3　分布に乱れを作る／ Step 4　境界処理を行う／ Step 5　*u*、*v* の分布を更新する／ Step 6　メッシュで分布を立体的に表示する／ Step 7　更新を繰り返し、最後にメッシュを表示する

付録　177

1　Grasshopper のコンポーネントを作る　178

2　巻貝の Grasshopper コンポーネント　182

3　Python 便利な関数リスト　185

Index　189

装丁　坂 哲二（BANG! Design, inc.）
イラスト　いとひろ（P.96、114、124、134、144、164）
樽井文花（上記以外すべて）

Rhinoceros for Macで Pythonスクリプトを始める方法

この本は、読者のみなさんがWindows版のRhinocerosを使っていることを前提としていますが、Mac版のRhinoでも同様のプログラミングをすることができますので、その手がかりについて少々触れておくことにします。

1. エディタのインストール

Rhinoのほかに、別途、任意のエディタをインストールする必要があります。Atom text editor が使いやすくて便利なのでおすすめします。Atom text editorは、http://atom.ioからダウンロードできます。まず、Mac用のAtom text editorをダウンロードして、インストールします。インストールが済んだらAtomを起動して、メニューバーのAtomというメニューから「Install Shell Commands」を選択します。次に、キーボードで［command］+［,］を押して「Settings」を起動し、左側の「install」ボタンをクリックします。替わって表示される画面で「Packages」ボタンをクリックして、「Search Packages」ボックスに`rhino-python`とタイプし、［Enter］キーを押します。その下に表示されるリストの先頭にrhino-pythonパッケージが現れるので「install」ボタンをクリックします。これでエディタの準備は完了です。

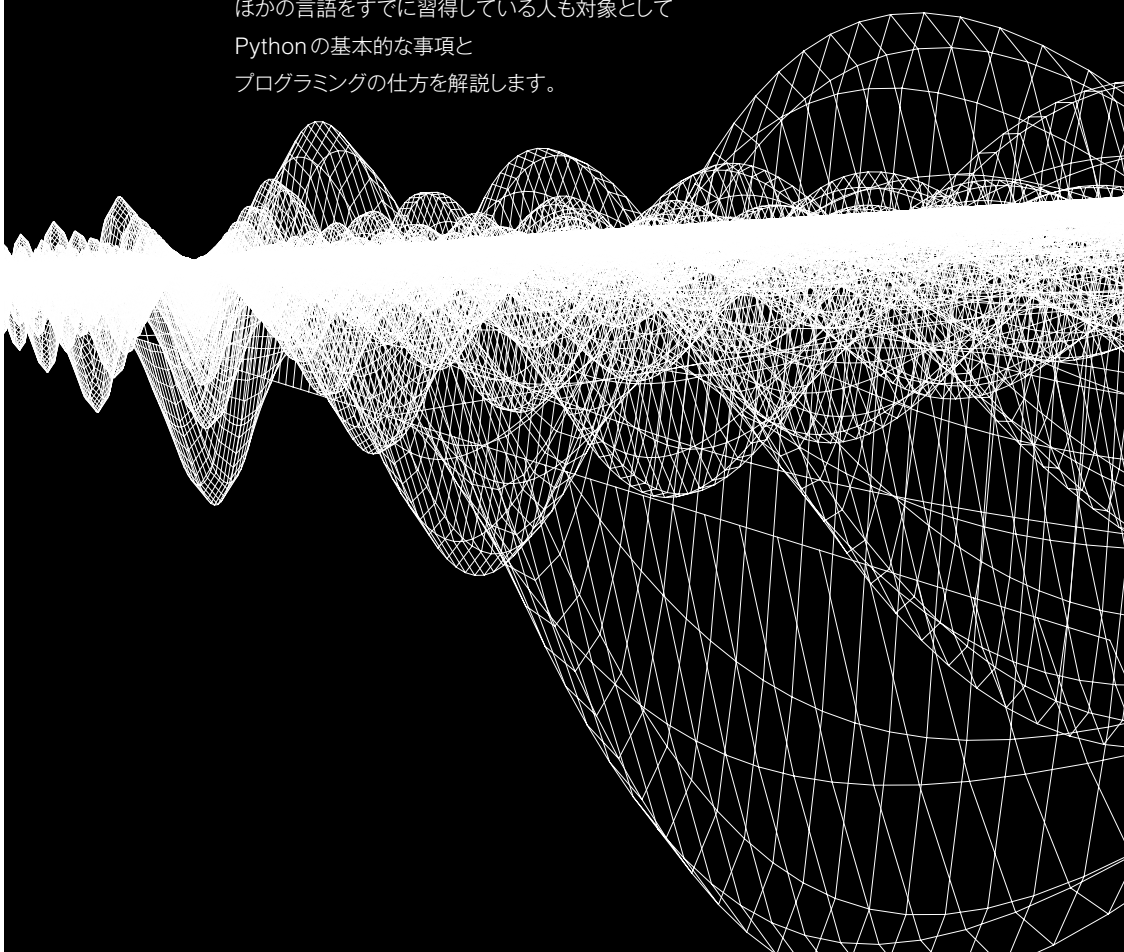
2. 編集と実行

Rhinoのコマンドエリアで`StartAtomEditorListener`コマンドを実行します。Atomを起動してPython スクリプトを編集し、「.py」という拡張子をつけてファイルを保存します。編集が完了したら、Atomをアクティブにしたままキーボードで［Control］+［alt］+［r］キーを押すと、Rhinoにファイルが送られて実行が始まります。

Chapter 1

Pythonプログラミングの基礎

Pythonは数あるプログラミング言語の一つです。
わが国においてCやC++、Javaなどに比べるとあまり知られていないかもしれませんが、
海外ではさまざまなアプリケーションへの組み込み、
学術計算などに多く用いられていて、利用できるライブラリも数多く存在しています。
また、プログラミング初心者の学習にも広く用いられています。
これは、Pythonで書かれたプログラムが読みやすいという特徴によるものでしょう。
ここでは、プログラミングをはじめて学ぶ人はもちろんのこと、
ほかの言語をすでに習得している人も対象として
Pythonの基本的な事項と
プログラミングの仕方を解説します。



1 ビルトインオブジェクト

Pythonのプログラムでは、処理の対象となるすべてのものが**オブジェクト**と呼ばれます。処理の対象とは、**数値**や**文字列**などです。例えば、プログラムでは与えられた数値の合計を計算するとか、与えられた文字列の中から特定の文字を探し出すなどといった処理をするわけです。数値も文字もデータとして扱うことができますが、それぞれ異なる特徴がありますね。数値と文字のような違いを、プログラムでは型（タイプ）の違いとして区別しています。Pythonには、あらかじめさまざまな機能を備えた何種類ものオブジェクトが用意されています。これらを総称して**ビルトインオブジェクト**と呼びます。ビルトインオブジェクトには、数値、文字列のほかに、**リスト**、**ディクショナリ**、**タプル**、**ブーリアン**などがあります（図1）。

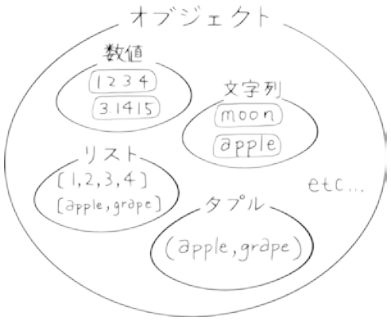


図1 オブジェクトの例

1. 数値を作るには、例えば、`1234`、`3.14159`、`999L`、`3 + 4j`のように書きます。これらは、数値リテラルと呼ばれます。`1234`や`3.14159`は日常で使う数字の書き方と同じです。`999L`は長整数の一例で、末尾に`L`を付けると桁数をいくらでも長くできます。また、`3 + 4j`は複素数を表現するリテラルの一例で、虚数部に`j`を付けます。
2. 文字列を作るには、例えば、`'moon'` `'apple'` `'book'` `'April'`のようにクオーテーション（'）またはダブルクオーテーション（"）で囲むという文字リテラルを使用します。
3. リストを作るには、角括弧`[ ]`で囲んで`[1, 2, 3, 4]`や`['apple', 'grape', 'lemon', 'orange']`のように、カンマ（`,`）で要素間を区切って表します。
4. ディクショナリを作るには、中括弧`{ }`で囲んで、`{39:'Italy', 49:'Germany', 66:'Thailand'}`のように書きます。コロン（`:`）で結ばれた左側は**キー**、右側は**値**と呼ばれます。
5. タプルを作るには、丸括弧`( )`で囲んで、`('apple', 'grape', 'lemon', 'orange')`のように書きます。
6. ブーリアンは真か偽を意味する`True`と`False`しかありません。ブーリアンを作るには、`True`または`False`を代入します。

数値

数値の主なものは、整数（小数点のない数値）、浮動小数点数（小数部分のある数値）です。数値は、普通の数学演算に使えます。例えば、加算なら`+`の記号を、乗算なら`*`を、累乗なら`**`を使います。


```
123 + 321, 1.25*4, 2**16
```

などは、それぞれ123と321の和、1.25と4の積、2の16乗を意味します。同様に、

```
444 - 321, 5 / 1.25
```

などは、444から321を引いた差、5を1.25で割った商を意味します。さらに、**mathモジュール**と呼ばれるプログラムファイルをインポートしておけば、用意されている円周率πを意味する`math.pi`も使うことができます。また、**randomモジュール**をインポートしておけば、`random.random()`のようにして乱数を生成することもできます。

文字列

文字列は複数の文字の並びのことで、**シーケンス**と呼ばれるPythonのオブジェクトの一種です。シーケンスというのは、オブジェクトを一定の順序に並べたもので、要素となるオブジェクトの順序は常に変わりません。位置を指定して、要素を抽出したり、追加したりすることができます。例えば、`'apple'`という文字列の先頭の`a`を取り出したり、末尾に`s`を付け加えて`'apples'`という文字列にしたりすることができます。

リスト

リストは、次に示す例のように角括弧`[ ]`で囲んで表記します。オブジェクトを一定の順序に並べたもので、シーケンスの一種です。その要素は、どのような型でもかまいません。

```
[1, 2, 3, 4]
```

この例では、整数が順に4つ並んでいます。文字列とは違って、要素を追加したり削除したりすることができます。このように変更できる性質を持ったオブジェクトを**可変性**のオブジェクトといいます。例えば、先頭に0を付け加えて`[0, 1, 2, 3, 4]`とすることもできますし、また末尾に5を追加して`[1, 2, 3, 4, 5]`とすることもできます。3番目の要素を0に置き換えて`[1, 2, 0, 4]`としたり、途中に9を挿入して`[1, 2, 9, 3, 4]`とすることもできます。

ディクショナリ

ディクショナリは、中括弧`{ }`で囲んで表記します。リストと同じようにオブジェクトの集合ですが、その要素の並ぶ順序は一定ではなく、その代わりに個々の要素にそれぞれのキーが付いています。

```
{81: 'Japan', 49: 'Germany', 66: 'Thailand', 39: 'Italy' }
```

この例では、81、49、66、39がキーであり、'Japan' 'Germany' 'Thailand' 'Italy'は値と呼ばれます。特定の要素にアクセスするには、インデックスではなく、キーを手がかりにします。また、ディクショナリは上書き可能な可変性のオブジェクトであり、リストと同様に必要に応じて追加や削除が可能です。

タプル

タプルは、次に示す例のように丸括弧( )で囲んで表記します。要素の型は何であってもかまいません。

```
('apple', 'grape', 'lemon', 'orange')
```

リストと似ていますが、**不変性**のオブジェクトである点が違います。つまり、上書きができないのです。タプルは、リストと比べると操作が自由にできないのですが、逆に大規模なプログラム中などでは、要素の変更ができないということで整合性が確保されるということが重要な意味を持つ場合があるのです。

ブーリアン

ブーリアンは、True（真）またはFalse（偽）のどちらかの値をとるため、真偽型と呼ばれることもあります。次の例は、変数aにTrueを、bにFalseを代入しています。

```
1        a = True
2        b = False
```

次のような論理演算の結果は、TrueまたはFalseのどちらかの値をとることになります。論理演算については後で述べますが、x < yはxがyより小さいかを、またx == yはxとyが等しいかどうかを判定します。この例では、cにはTrue、dとeにはFalseが代入されるでしょう。

```
1        x = 12
2        y = 18
3        c = x < y
4        d = x > y
5        e = x == y
```

これらの操作の詳細については、次の「変数」について学んだ後で説明することにします。

2 変数

変数の役割

変数は、プログラム中で使用するオブジェクトの入れ物のような役割を果たします。Pythonの変数がほかのプログラミング言語と違うのは、あらかじめ変数を宣言しなくてもいいという点です。例えばC言語などでは、整数の入れ物としてaなどという名前の変数を宣言しておいて、その後にオブジェクトを入れる、すなわち**代入**という手順をとります。しかし、Pythonではその必要がなく、はじめに値が代入された時点で変数が作成されるのです。

```
a = 3
```

と書くと、aは変数、=は代入、3は整数型オブジェクトを意味して、変数aに整数3が格納されます（図2）。このように、aが変数であることをあらかじめ明記しなくても、さらにaが整数型オブジェクトの入れ物であることを宣言しなくても、3を代入するという処理が行われた時点でaという名前の変数が作成されるのです。変数を使って式を書くと、演算にはその変数に代入された値が使用されます。ですから、先ほどの例につづけて、

```
b = a + 2
```

とすれば、変数aに格納された3と2の和が計算されて変数bには5が代入されます（図3）。

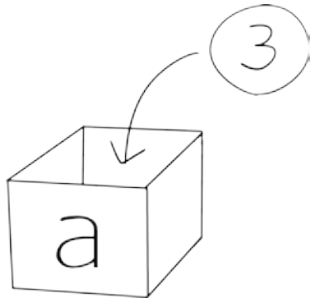


図2　変数aに整数3を代入する

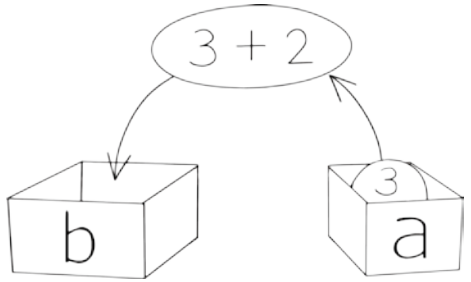


図3　3を代入したaと2の和5がbに代入される

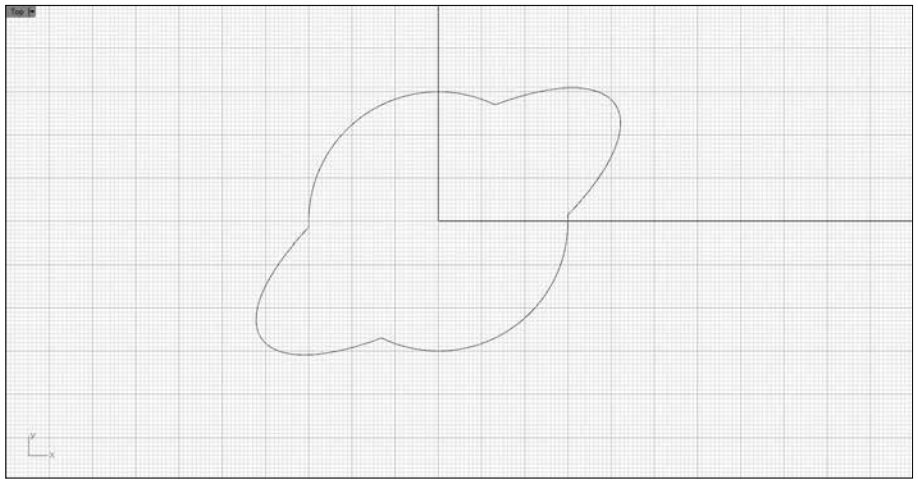
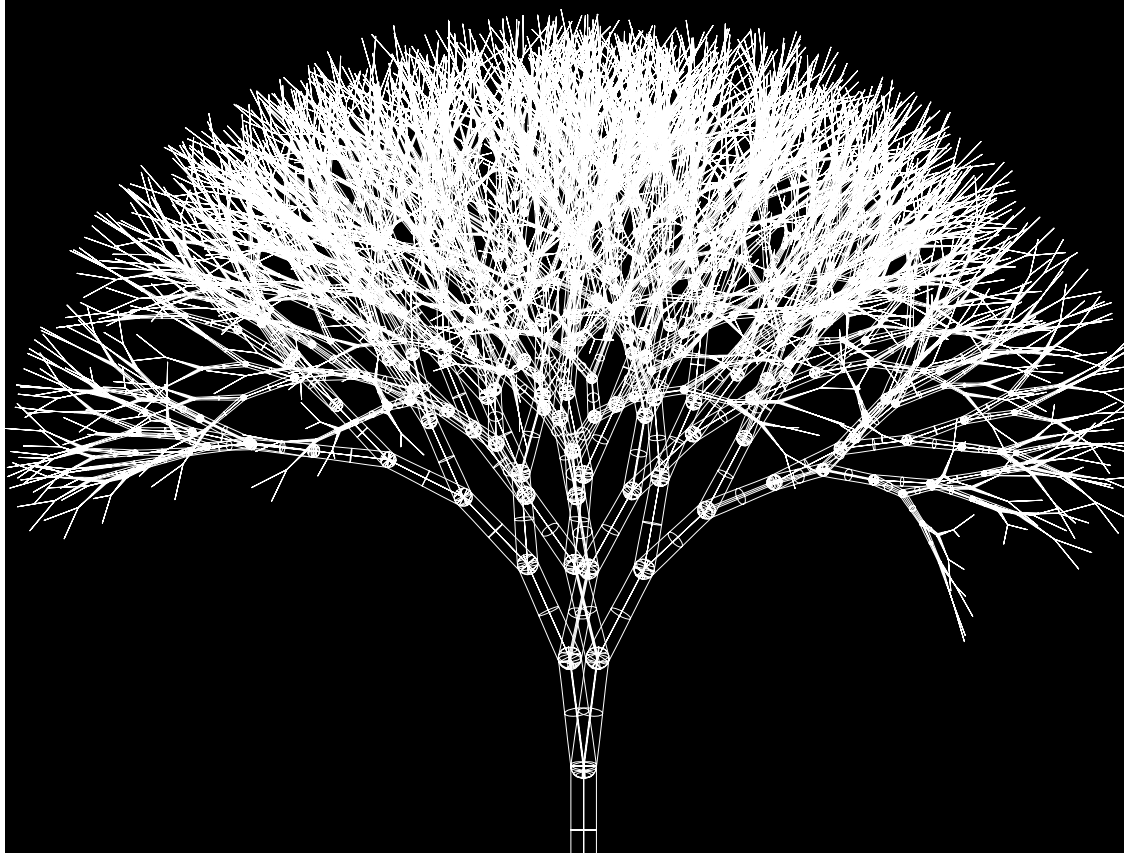


図42　閉曲線の結合によって生成された図形

Chapter 3

コンピュテーショナル・デザイン

Chapter1とChapter2で学んだプログラミングのテクニックを使って、6つの問題に挑戦します。
1つ目はサボテンです。ここでは、イメージした形をプログラミングを通して表現する方法を学びます。
2つ目は貝殻です。イメージを数式に置き換えて、プログラミングへつづけます。
3つ目は樹木です。再帰というプログラミンングに特有な方法を学びます。
再帰と自然界のさまざまな事象との関係を知ることができるでしょう。
4つ目はワッフリングです。立体を平面の組み合わせで表現することについて学びます。
5つ目は最適化を扱います。最適化という手法、
なかでも発見的最適化を使って形状を見つける方法を学びます。
6つ目はチューリング・パターンです。
方程式から形状が生成されるという一例を知ることができるでしょう。



1 イメージした形とプログラム

イメージした形を紙に鉛筆でスケッチするように、プログラミングでその形を画像にしてみましょう。紙と鉛筆のスケッチとはちがって、パラメータを変更したり、プログラム・コードの数行を書き換えたりするだけで何度もイメージを修正することが可能です。また、出来上がった画像のデータを3Dプリンターに送って、実際に手にして眺めたり触ったりすることができる物質として出力することも可能になります。

ここで取り上げるイメージは図1のようなサボテンです。サボテンにもいろいろな種類がありますが、球形のサボテンを選びました。球形の表面に子午線方向に伸びる棘（とげ）の尾根が並び、尾根と尾根の間は谷となっています。そして、この尾根に鋭い棘が数本ずつ束となって規則正しく生えています。棘の根元には刺座（しざ）またはアレオーレと呼ばれる器官がありますが、これは省略します。尾根の数は種類によってまちまちで、数が少なく比較的高い尾根と深い谷となっているものや、数が多くて尾根と谷の高さにあまり差のないものまでさまざまです。

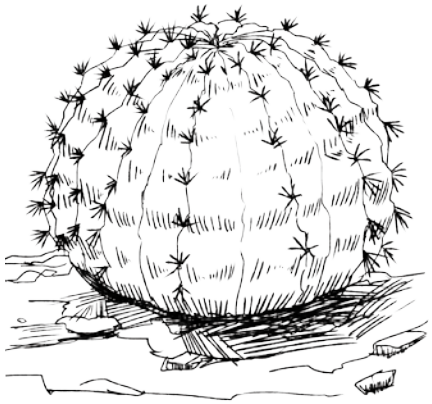


図1 球形のサボテン

プログラミングのアウトライン

プログラミングのアウトラインを1〜7に示します。おおまかなあらすじを把握してから始めましょう。

1. 尾根の数、高さ、棘の数、高さ、太さなどのパラメータを決める。
2. 尾根の形を作るための曲線を決める。
3. 尾根を適切な位置に配置する。
4. 断面のカーブを補間して曲面を作成するコマンドLoftによってサボテンの本体を作るとし、そのための断面を生成する。
5. 断面をロフト（Loft）して本体を作る。
6. 棘は円錐を組み合わせて作るとし、原点に棘の基本形を作成する。
7. 尾根曲線の各点に棘の基本形を適切な向きに貼り付ける。

Step 1 パラメータを設定する

尾根の数、高さ、棘の数、高さ、太さなどのパラメータを決めましょう。尾根の数、高さをそれぞれ`n`、`a`、棘の数、高さ、太さはそれぞれ`m`、`sh`、`sr`とします。長さの単位はmm（ミリメートル）を使います。

```
1       import rhinoscriptsyntax as rs
2       import math as ma
3
4       n = 12
5       a = 4
6       m = 6
7       sh = 10
8       sr = 0.5
```

```
1       rhinoscriptsyntaxモジュールをインポートし、rsという略称を付ける。
2       数学のモジュールmathをインポートし、maという略称を付ける。
4       尾根の数をnとし、12を代入する。
5       尾根の高さをaとし、4を代入する。
6       棘の数をmとし、6を代入する。
7       棘の高さをshとし、10を代入する。
8       棘の底面における半径をsrとし、0.5を代入する。
```

Step 2 尾根曲線を決める

尾根曲線を決定しましょう。このプログラムでは、*x*-*z*平面上（Front）で尾根曲線を根元の方から描いて*z*軸上に終点を置くことを前提にします。ですから、Rhinoのコマンドを使って、Front画面で図2のように尾根曲線を根元から頂部に向かって描いて準備します。サボテンはこの曲線を*z*軸のまわりに回転して作ります。これから作るプログラムでは、直径も高さも100mm程度の大きさを想定しています。別のサイズにしたいなら、棘のサイズなどのパラメータを調節してください。

まず、準備した曲線をプログラム中に取得します。Rhinoのドキュメント上に描かれた曲線を選択して`curve`と名付けるには`rs.GetObject()`関数を使います。

```
10      curve = rs.GetObject("Select a curve", 4)
```

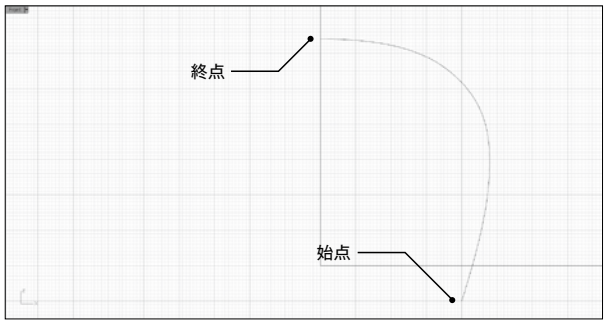


図2 RhinoのFront画面に尾根曲線を描いて準備する